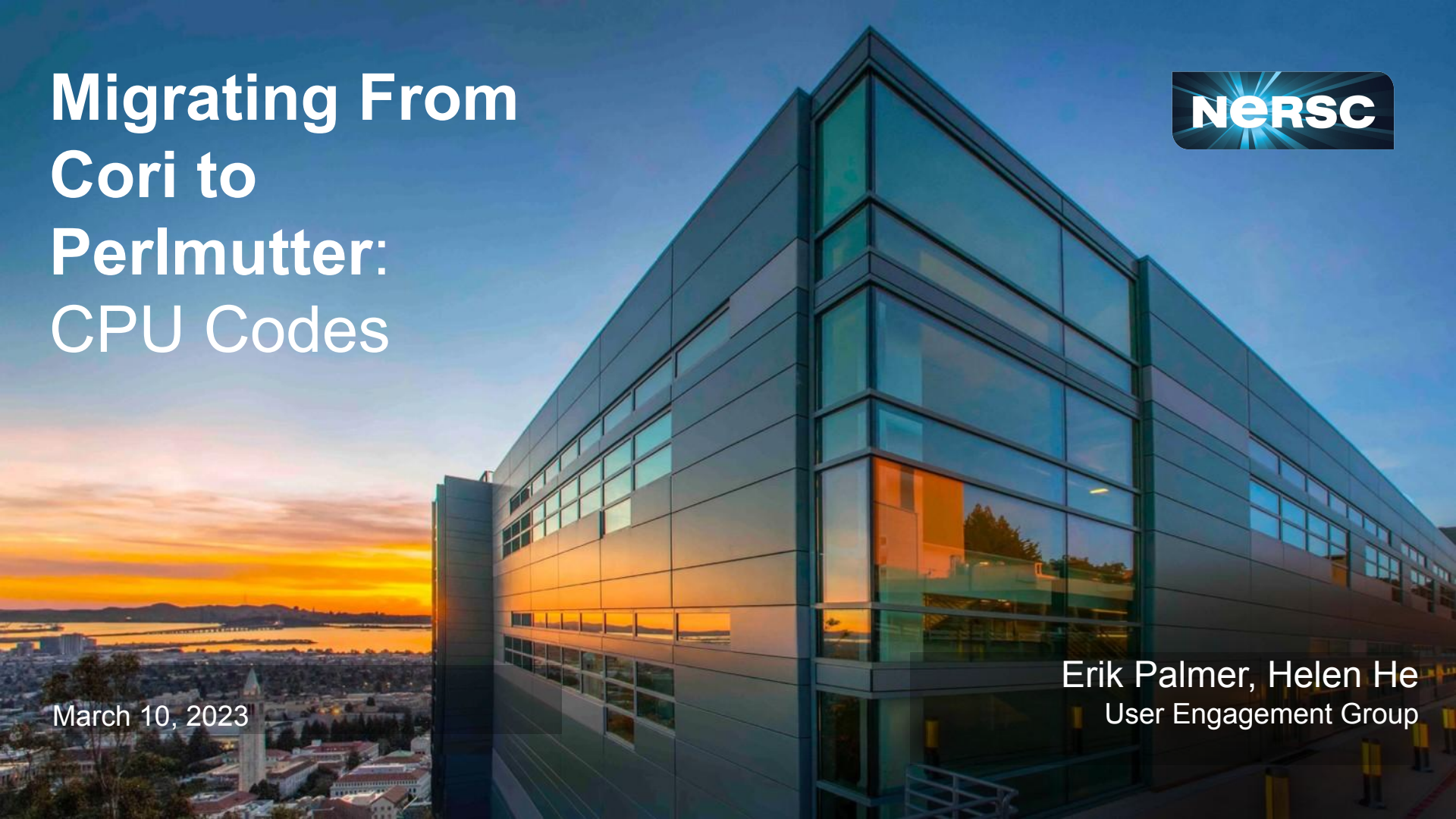
Migrating From Cori to Perlmutter: CPU Codes

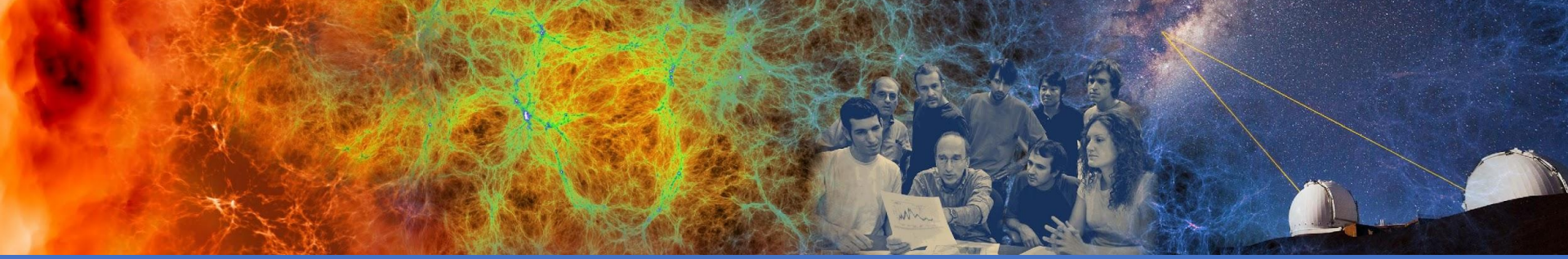


March 10, 2023

Erik Palmer, Helen He
User Engagement Group

Outline

- Modules
- Programing Environments
- Makefiles and CMake tips
- Example Code Compilation
- Job Scripts & Affinity



Modules: Loading Preinstalled Software

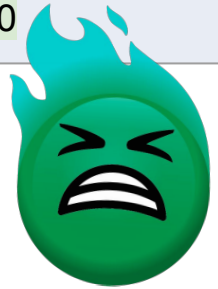
`module ~~avail~~ spider`

Modules Loaded at Login

Modules Loaded by Default:

```
epalmer@perlmutter:login21:~> $ module list
 1) craype-x86-milan                      7) cray-libsci/23.02.1.1          13) xalt/2.10.2
 2) libfabric/1.15.2.0                    9) cray-mpich/8.1.24              14) Nsight-Compute/2022.1.1
 3) craype-network-ofi                    7) craype/2.7.19                  15) Nsight-Systems/2022.2.1
 5) xpmem/2.5.2-2.4_3.30__gd0f7937.shasta 6) gcc/11.2.0                     16) cudatoolkit/11.7
11) PrgEnv-gnu/8.3.3                      4) perftools-base/23.02.0         17) craype-accel-nvidia80
 8) cray-dsmml/0.2.2                     12) cpe/23.02                      18) gpu/1.0
```

- CPU Architecture
- Default Programming Environment and Compiler
- GPU Architecture, CUDA-Aware MPI, GPU Profilers



CUDA-Aware MPI by Default!

Default Modules for CPU-only Code

For CPU-only code we recommend:
`module load cpu`

```
epalmer@perlmutter:login21:~> $ module load cpu
epalmer@perlmutter:login21:~> $ module list
```

1) craype-x86-milan
2) libfabric/1.15.2.0
3) craype-network-ofi
4) xpmem/2.5.2-2.4_3.30__gd0f7937.shasta
5) PrgEnv-gnu/8.3.3
6) cray-dsmml/0.2.2
7) cray-libsci/23.02.1.1
8) cray-mpich/8.1.24
9) craype/2.7.19
10) gcc/11.2.0
11) perftools-base/23.02.0
12) cpe/23.02
13) xalt/2.10.2
14) cpu/1.0

- CPU Architecture
- Default Programming Environment and Compiler
- Configured for CPU-only MPI

CUDA-Aware MPI turned off!

Modules with Lmod

Most Common

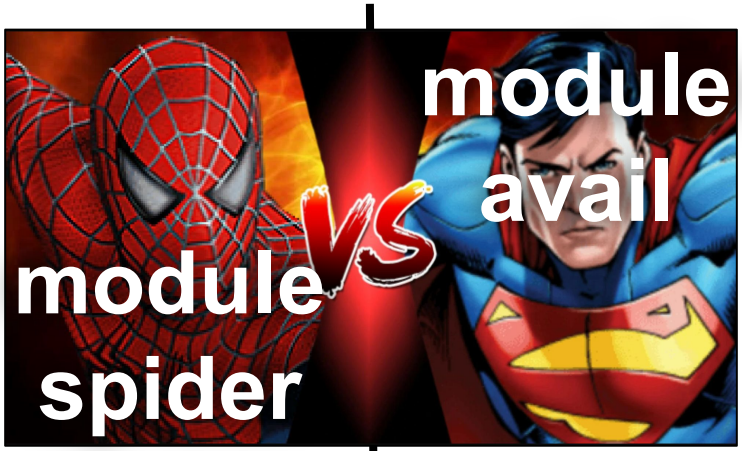
- `module list`
- `module load/unload`
- `module spider`
- `module swap`
- `module show`

Cool Tricks

- `module --redirect spider -r . | grep <string>`
- `ml -t`

More information: `man module` or https://docs.nersc.gov/environment/lmod/

Hierarchical Structure of Modules on Perlmutter



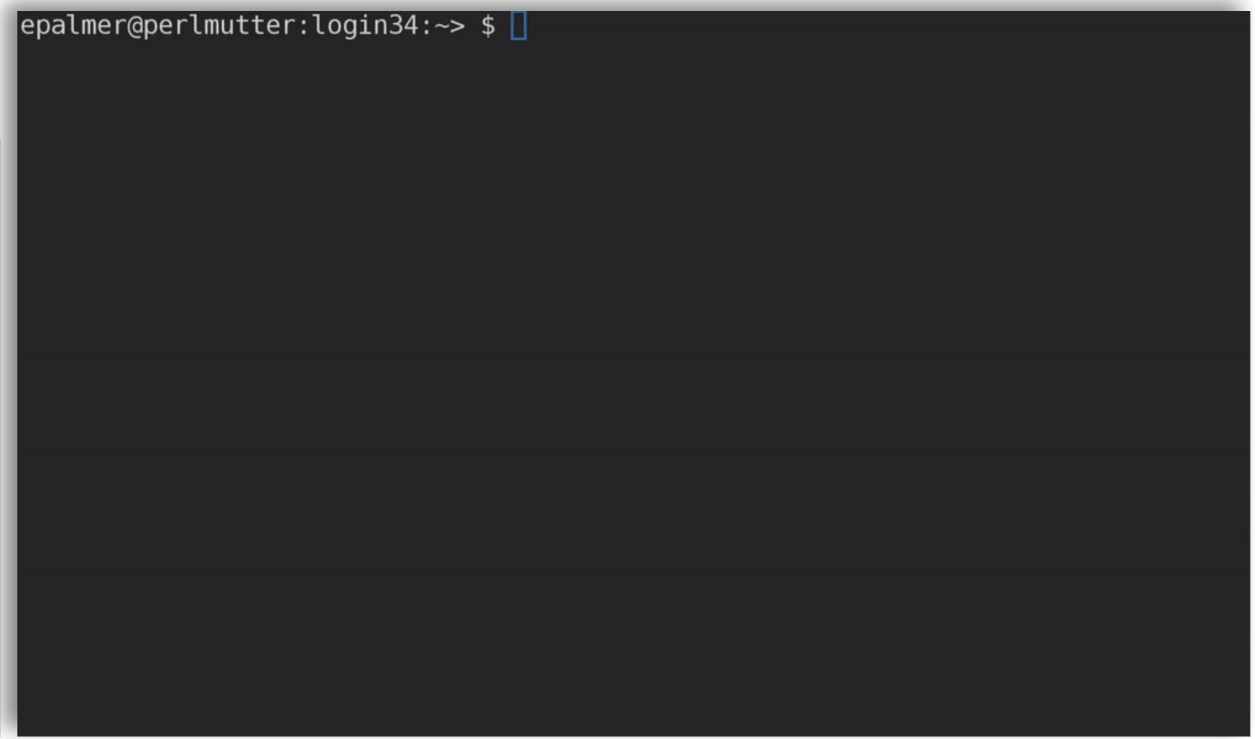
- Searches all modules without regard for hierarchy
- Displays only modules that can currently be loaded

Module Example: Searching for `cray-netcdf`

No Longer Recommended

- `module avail`

  *Only shows packages that can be loaded into the current module environment (due to hierarchy) – **use `module spider` instead**



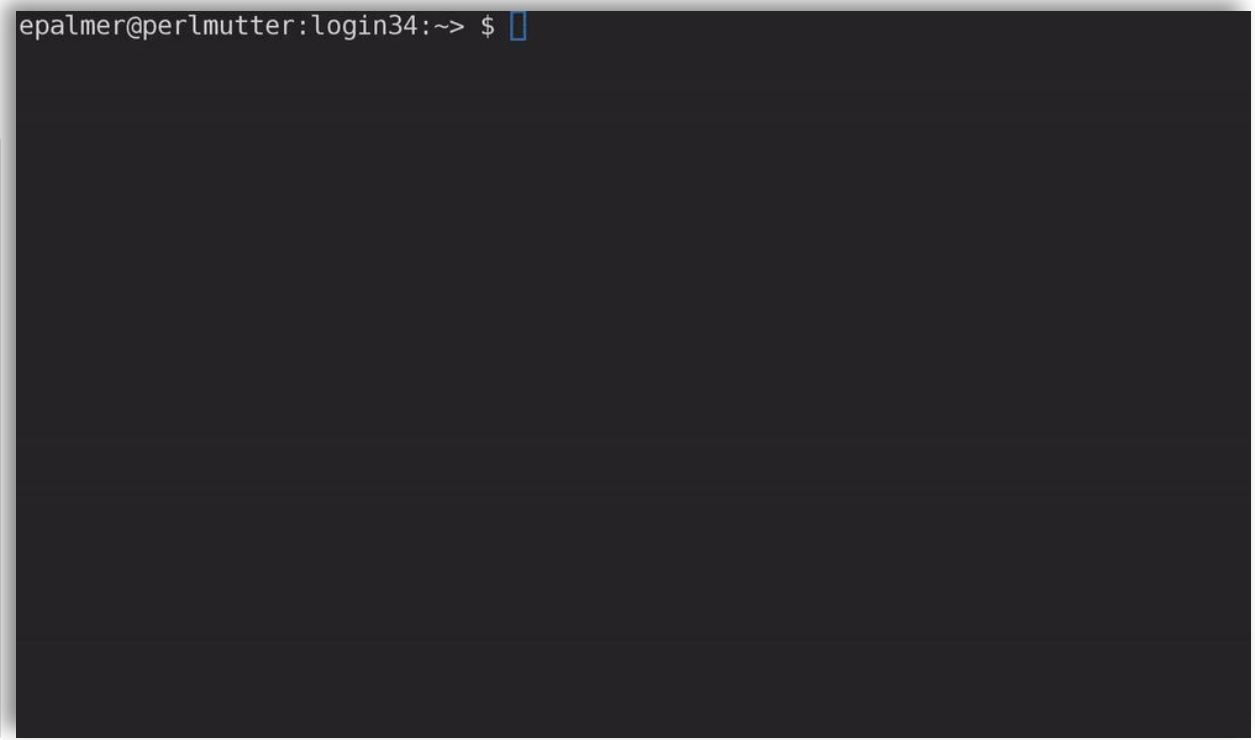
More information: https://docs.nersc.gov/environment/lmod/

Module Example: Searching for `cray-netcdf`

No Longer Recommended

- `module avail`

  *Only shows packages that can be loaded into the current module environment (due to hierarchy) – **use `module spider` instead**

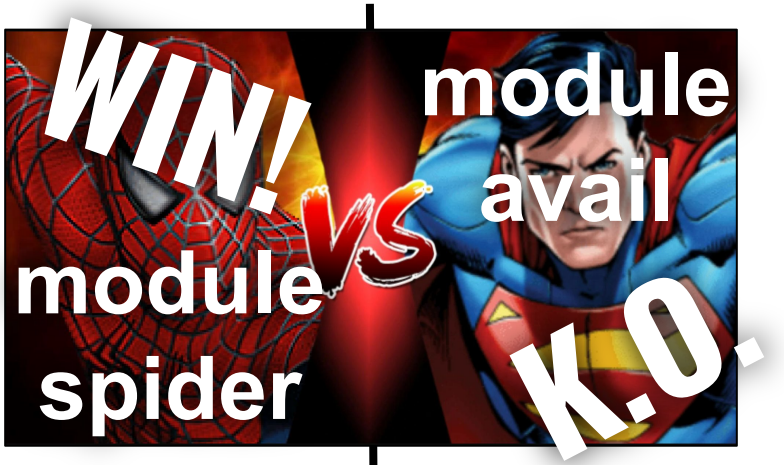


More information: https://docs.nersc.gov/environment/lmod/

List of commands from previous slide

1. `module list`
2. `module load cray-netcdf`
3. `module show cray-netcdf`
4. `module avail`
5. `module spider cray-netcdf`
6. `module spider cray-netcdf/4.8.1.5`
7. `module load cray-hdf5`
8. `module load cray-netcdf`

New Hierarchical Structure of Modules on Perlmutter

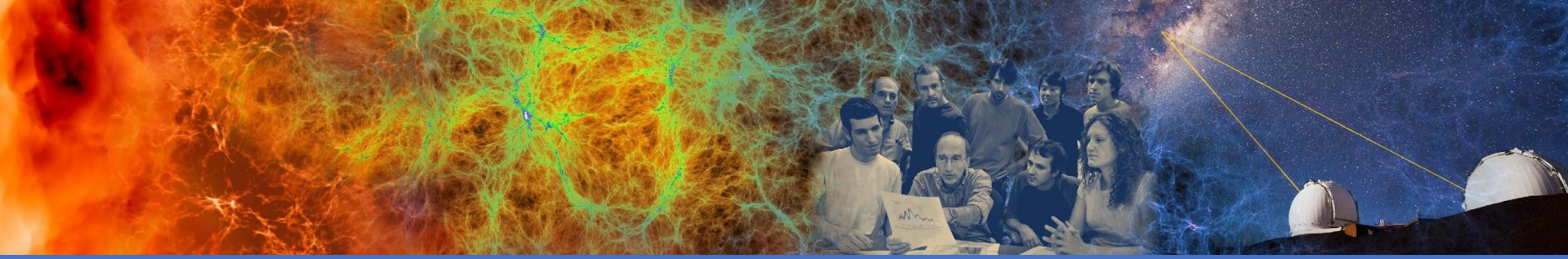


- Searches for modules without regard for hierarchy
- Displays only modules that can currently be loaded

Loading Modules Modifies Your Environment

```
epalmer@perlmutter:login06:~> $ module show cray-hdf5
-------------------------------------------------------------------------------------------------------------------------------
   /opt/cray/pe/lmod/modulefiles/compiler/gnu/8.0/cray-hdf5/1.12.2.3.lua:
-------------------------------------------------------------------------------------------------------------------------------
family("hdf5")
conflict("PrgEnv-pathscale")
conflict("cray-hdf5")
conflict("cray-hdf5-parallel")
help([[Release info:  /opt/cray/pe/hdf5/1.12.2.3/release_info]])
whatis("The HDF5 Technology suite includes tools and applications for managing, manipulating, viewing, and analyzing
data in the HDF5 format.")
prepend_path("PATH","/opt/cray/pe/hdf5/1.12.2.3/bin")
prepend_path("PKG_CONFIG_PATH","/opt/cray/pe/hdf5/1.12.2.3/gnu/9.1/lib/pkgconfig")
prepend_path("PE_PKGCONFIG_LIBS","hdf5_hl:hdf5")
setenv("PE_HDF5_PKGCONFIG_LIBS","hdf5_hl:hdf5")
prepend_path("PE_FORTRAN_PKGCONFIG_LIBS","hdf5hl_fortran:hdf5_fortran")
setenv("PE_HDF5_FORTRAN_PKGCONFIG_LIBS","hdf5hl_fortran:hdf5_fortran")
prepend_path("PE_CXX_PKGCONFIG_LIBS","hdf5_hl_cpp:hdf5_cpp")
setenv("PE_HDF5_CXX_PKGCONFIG_LIBS","hdf5_hl_cpp:hdf5_cpp")
setenv("CRAY_HDF5_DIR","/opt/cray/pe/hdf5/1.12.2.3")
setenv("PE_HDF5_DIR","/opt/cray/pe/hdf5/1.12.2.3")
setenv("CRAY_HDF5_VERSION","1.12.2.3")
setenv("CRAY_HDF5_PREFIX","/opt/cray/pe/hdf5/1.12.2.3/gnu/9.1")
setenv("HDF5_DIR","/opt/cray/pe/hdf5/1.12.2.3/gnu/9.1")
setenv("HDF5_ROOT","/opt/cray/pe/hdf5/1.12.2.3/gnu/9.1")
prepend_path("CRAY_LD_LIBRARY_PATH","/opt/cray/pe/hdf5/1.12.2.3/gnu/9.1/lib")
prepend_path("MODULEPATH","/opt/cray/pe/lmod/modulefiles/hdf5/gnu/8.0/cray-hdf5/1.12.2")
```

Path Changes
Environment Variables
Other Info



Programming Environments: Configuring Compilers and Linking Libraries

Programming Environments
Control Compilers and Libraries on Perlmutter

Language	Wrapper	PrgEnv-gnu (default)	PrgEnv-nvidia	PrgEnv-cray
C++	CC	g++	nvc++	crayCC (Clang)
C	cc	gcc	nvc	craycc (Clang)
Fortran	ftn	gfortran	nvfortran	crayftn
MPI	-	cray-mpich	cray-mpich	cray-mpich
More info:		module show PrgEnv-gnu	module show nvidia	module show cce

Programming Environments
Control Compilers and Libraries on Perlmutter

Language	Wrapper	PrgEnv-gnu (default)	PrgEnv-nvidia	PrgEnv-cray
C++	**CC**	g++	nvc++	crayCC (Clang)
C	**cc**	gcc	nvc	craycc (Clang)
Fortran	**ftn**	gfortran	nvfortran	crayftn
MPI	-	cray-mpich	cray-mpich	cray-mpich
More info:		module show PrgEnv-gnu	module show nvidia	module show cce

Switching Programming Environments

To switch programming environments (PrgEnv) use:

```
module load PrgEnv-*
```

For example,

if I am in `PrgEnv-gnu` and want to switch to `PrgEnv-cray`, type:

```
module load PrgEnv-cray
```

Suggested Practice: Compiler Wrappers

Cray provides wrappers (**`CC`**, **`cc`**, and **`ftn`**) to the corresponding compilers for each PrgEnv. These wrappers incorporate many flags and features, including `cray-mpich`, the recommended MPI.

```
epalmer@nid005015:~/Training> gcc helloworld_openmp.c -fopenmp -o hello
```

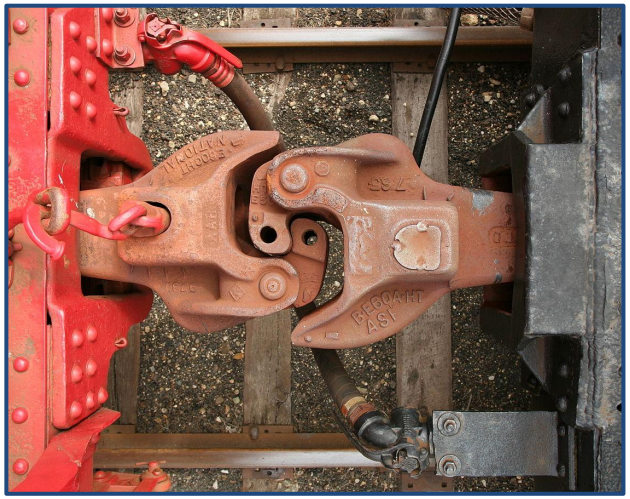
```
epalmer@nid005015:~/Training> cc -craype-verbose helloworld_openmp.c -fopenmp -o hello

gcc -march=znver3 -D__CRAY_X86_MILAN -D__CRAY_NVIDIA80
-D__CRAYXT_COMPUTE_LINUX_TARGET -D__TARGET_LINUX__ helloworld_openmp.c
-fopenmp -o hello -Wl,-rpath=/opt/cray/pe/gcc-libs -Wl,-Bdynamic
-I/opt/cray/pe/mpich/8.1.24/ofi/gnu/9.1/include
-I/opt/cray/pe/libsci/23.02.1.1/GNU/9.1/x86_64/include
-I/opt/nvidia/hpc_sdk/Linux_x86_64/22.5/cuda/11.7/nvvm/include
-I/opt/nvidia/hpc_sdk/Linux_x86_64/22.5/cuda/11.7/extras/CUPTI/include
-I/opt/nvidia/hpc_sdk/Linux_x86_64/22.5/cuda/11.7/extras/Debugger/include …(and more)
```

Automatic Links Provided By The Wrappers

- Depending on modules loaded, compiler wrappers **automatically** link:

  MPI, LAPACK, BLAS, ScaLAPACK, and more.

- Cray modules, such as `cray-hdf5`, `cray-fftw`, etc. are also linked automatically by the compiler wrappers when loaded into the user environment.



Note: Several scientific libraries such as, LAPACK, ScaLAPACK, and BLAS, are included in `cray-libsci`. For more information use: `man libsci`.

Modules Link Dynamically by Default

- Many modules prepend the `LD_LIBRARY_PATH` or `CRAY_LD_LIBRARY_PATH`, and have their shared libraries dynamically linked.
  e.g.

```
module load gsl
CC gsl_test.cpp -lgsl -lgslcblas -o gsl_test
```

- If you are compiling your own shared libraries, consider using the option, `-Wl,-rpath=<library path>`. Cray wrappers build dynamically linked executables by default.
- On Perlmutter static compilation with `-static` or `CRAYPE_LINK_TYPE=static` can fail and is not supported.

More info: https://docs.nersc.gov/development/compilers/wrappers/

Other Good-to-Know Compiler Settings for CPU

GNU	Cray	Nvidia	Description/ Comment
-O0	-O0	-O1	Default Optimization Level
-Ofast	-Ofast, -flto	-O4, -fast	Aggressive Optimization (some may cause non-bit-identical output)
-fopenmp	-fopenmp	-mp	Enable OpenMP (not default)
-g, -O0	-g, -O0	-g (-O0 by default)	Debug
-v	-v	-v	Verbose

For more information, when in the corresponding PrgEnv type:

`man gcc/gfortran`	`man craycc/crayftn`	`man nvc/nvfortran`

https://docs.nersc.gov/development/compilers/

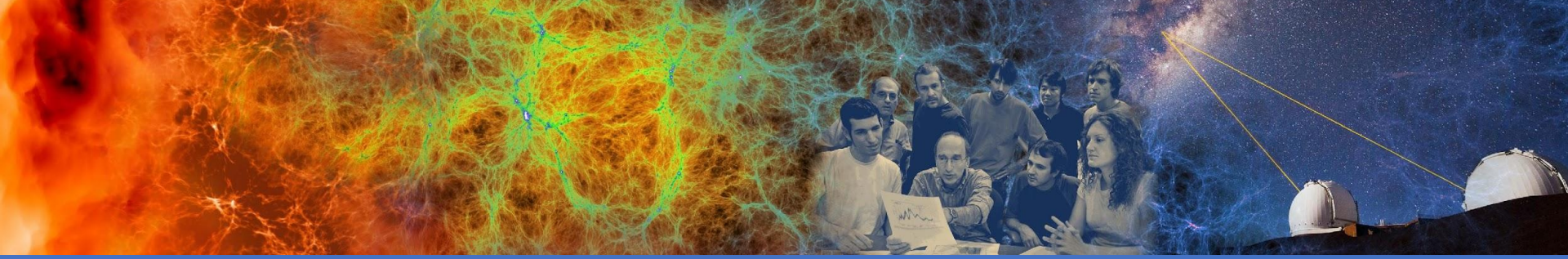
Three Quick Tips for Compiling Older Codes

Fortran:

1. For older codes, first try **`-fallow-argument-mismatch`** then if needed, we recommend the **`-std=legacy`** flag which includes additional modifications that reduce strictness.

C/C++:

2. Look for flags that reduce strictness, such as **`-fpermissive`**
3. **`-Wpedantic`** can warn you about lines that break code standards



CMake and Makefiles: Quick Tips on troubleshooting build systems

Manually Specify Cray Compiler Wrappers

Some build systems such as CMake or Makefiles may be coded to search for CC, CXX and FC environment variables.

In these cases, it is possible to specify the Cray compile wrappers by setting the environment variables in the following way:

```
CC=$(which cc) CXX=$(which CC) FC=$(which ftn)
```

Or at the configure step,

```
./configure CC=cc CXX=CC FC=ftn F77=ftn
```

More info: https://docs.nersc.gov/development/build-tools/autoconf-make/
https://docs.nersc.gov/development/build-tools/cmake/

Compiling with an Existing Makefile

Suppose you already have a Makefile

```
CC=gcc
CFLAGS=-I.
DEPS = hellomake.h

%.o: %.c $(DEPS)
	$(CC) -c -o $@ $< $(CFLAGS)

hellomake: hellomake.o hellofunc.o
	$(CC) -o hellomake hellomake.o hellofunc.o
```

Does not use the Cray Compiler Wrappers!

Makefile from: https://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/
More info: https://docs.nersc.gov/development/build-tools/autoconf-make/

Compiling with an Existing Makefile

Suppose yo… a Makefile

Edit so that, CC=cc

```
CC=gcc
CFLAGS=-I.
DEPS = hellomake.h

%.o: %.c $(DEPS)
    $(CC) -c -o $@ $< $(CFLAGS)

hellomake: hellomake.o hellofunc.o
    $(CC) -o hellomake hellomake.o hellofunc.o
```

Makefile from: https://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/
More info: https://docs.nersc.gov/development/build-tools/autoconf-make/

Compiling with an Existing Makefile

Suppose you already have a Makefile

```
CC=cc
CFLAGS=-I.
DEPS = hellomake.h

%.o: %.c $(DEPS)
    $(CC) -c -o $@ $< $(CFLAGS)

hellomake: hellomake.o hellofunc.o
    $(CC) -o hellomake hellomake.o hellofunc.o
```

Uses Cray Compiler Wrappers!

Makefile from: https://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/
More info: https://docs.nersc.gov/development/build-tools/autoconf-make/

CMake: Troubleshooting with ccmake

You can use the in-terminal gui ccmake to troubleshoot CMake builds.

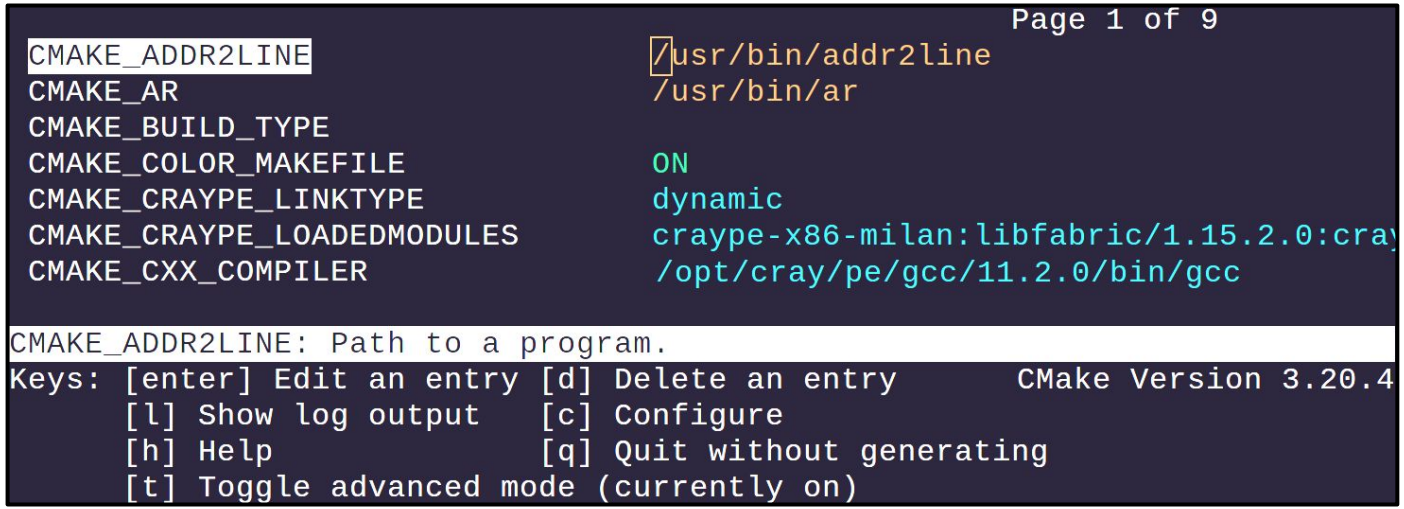
Starting from the directory, I follow a typical CMake build pattern:

```
epalmer>$ ls
CMakeLists.txt  openmp_helloworld.cpp  README.md
epalmer>$ mk build; cd build
epalmer>$ cmake ..
epalmer>$ ls
CMakeCache.txt  CMakeFiles  cmake_install.cmake  Makefile
epalmer>$ ccmake ..
```

This will bring up the ccmake interface.

CMake: Troubleshooting with ccmake

ccmake gui interface (advanced mode):



We can now inspect the options chosen by CMake.

More info: https://docs.nersc.gov/development/build-tools/cmake/

CMake: Troubleshooting with ccmake

ccmake gui interface (advanced mode):

```
                                                  Page 1 of 9
CMAKE_ADDR2LINE                  /usr/bin/addr2line
CMAKE_AR                         /usr/bin/ar
CMAKE_BUILD_TYPE
CMAKE_COLOR_MAKEFILE             ON
CMAKE_CRAYPE_LINKTYPE            dynamic
CMAKE_CRAYPE_LOADEDMODULES       craype-x86-milan:libfabric/1.15.2.0:cray
CMAKE_CXX_COMPILER               /opt/cray/pe/gcc/11.2.0/bin/gcc

CMAKE_ADDR2LINE: Path to a program.
Keys: [enter] Edit an entry [d] Delete an entry      CMake Version 3.20.4
      [l] Show log output   [c] Configure
      [h] Help              [q] Quit without generating
      [t] Toggle advanced mode (currently on)
```

Does not use the Cray Compiler Wrappers!

More info: https://docs.nersc.gov/development/build-tools/cmake/

CMake: Troubleshooting with ccmake

Inspecting another build or possibility:

More info: https://docs.nersc.gov/development/build-tools/cmake/

CMake: Troubleshooting with ccmake

Inspecting another build or possibility:

```
                                                    Page 1 of 9
CMAKE_ADDR2LINE                  /usr/bin/addr2line
CMAKE_AR                         /usr/bin/ar
CMAKE_BUILD_TYPE
CMAKE_COLOR_MAKEFILE             ON
CMAKE_CRAYPE_LINKTYPE            dynamic
CMAKE_CRAYPE_LOADEDMODULES       craype-x86-milan:libfabric/1.15.2.0:cray
CMAKE_CXX_COMPILER               /opt/cray/pe/craype/2.7.19/bin/CC

CMAKE_ADDR2LINE: Path to a program.
Keys: [enter] Edit an entry [d] Delete an entry      CMake Version 3.20.4
      [l] Show log output   [c] Configure
      [h] Help              [q] Quit without generating
      [t] Toggle advanced mode (currently on)
```

Uses Cray Compiler Wrappers!

More info: https://docs.nersc.gov/development/build-tools/cmake/

Manually Specify Cray Compiler Wrappers

Some build systems such as CMake or Makefiles may be coded to search for CC, CXX and FC environment variables.

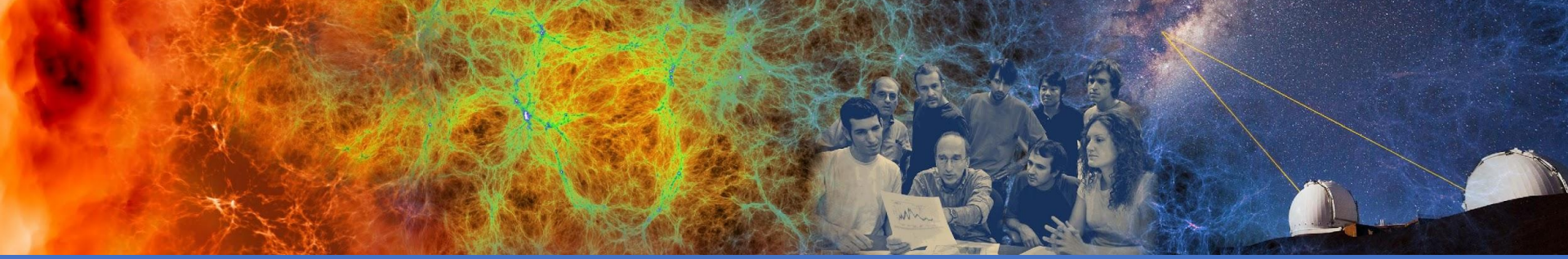
In these cases, it is possible to specify the Cray compile wrappers by setting the environment variables in the following way:

```
CC=$(which cc) CXX=$(which CC) FC=$(which ftn)
```

Or at the configure step,

```
./configure CC=cc CXX=CC FC=ftn F77=ftn
```

More info: https://docs.nersc.gov/development/build-tools/autoconf-make/
https://docs.nersc.gov/development/build-tools/cmake/



Building Applications: Cori codes should be recompiled for Perlmutter

Example CPU Code Compile with MPI and OpenMP

```
int main(int argc, char *argv[]) {
  int numprocs, rank, namelen;
  char processor_name[MPI_MAX_PROCESSOR_NAME];
  int iam = 0, np = 1;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Get_processor_name(processor_name, &namelen);

  #pragma omp parallel default(shared) private(iam, np)
  {
    np = omp_get_num_threads();
    iam = omp_get_thread_num();
    printf("Hello from thread %d out of %d from process
           %d out of %d on %s\n",
           iam, np, rank, numprocs, processor_name);
  }
  MPI_Finalize();
 }
```

Hellohybrid.c – contains both MPI and OpenMP commands.

Example from: https://rcc.uchicago.edu/docs/running-jobs/hybrid/index.html

Compile with MPI and OpenMP (CPU only)

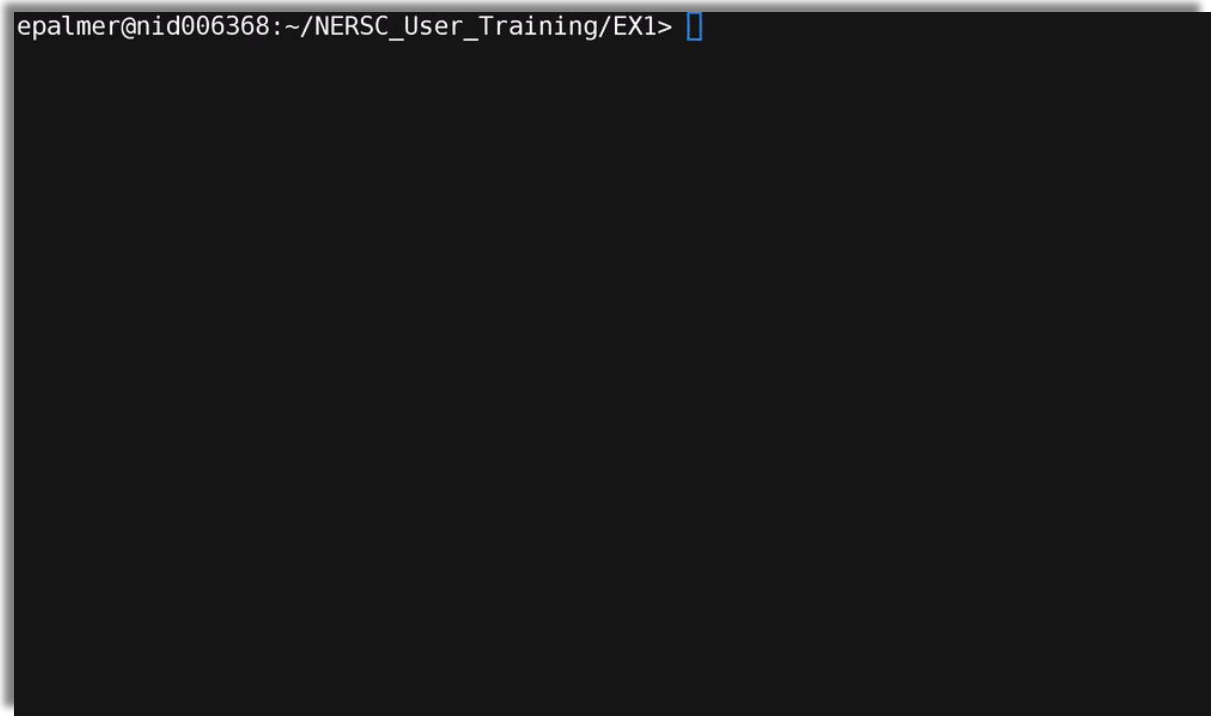
```
epalmer@nid006368:~/NERSC_User_Training/EX1> #In this example, we will compile a
 code with MPI and OpenMP
```

Modules Loaded:
PrgEnv-gnu

Compile line:

```
cc hellohybrid.c -fopenmp -o hellohybrid
```

Compile with MPI and OpenMP (CPU only)



Modules Loaded:
PrgEnv-gnu

Compile line:

```
cc hellohybrid.c -fopenmp -o hellohybrid
```

Compile with MPI and OpenMP (CPU only)

```
epalmer@nid006368:~/NERSC_User_Training/EX1>
```

Compilation with wrappers is very similar.* `-fopenmp`

Compile line:

```
cc hellohybrid.c -fopenmp -o hellohybrid
```

List of commands from previous slide

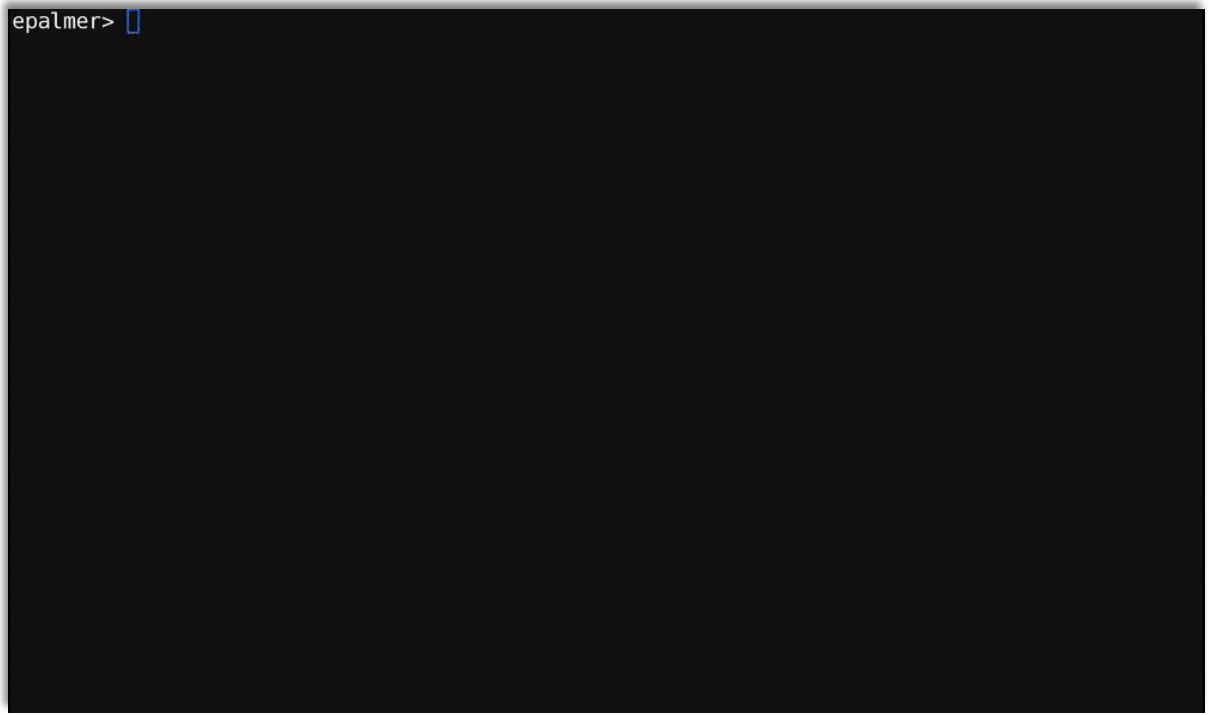
```
1. ls
2. module list
3. cc hellohybrid.c -fopenmp -o hellohybrid
4. export OMP_NUM_THREADS=2
5. export OMP_PROC_BIND=spread
6. export OMP_PLACES=threads
7. srun -n 4 -c 2 ./hellohybrid
```

*__Note__: #5 in the gif is incorrect, it should be `OMP_PROC_BIND=spread` not `OMP_PROC_BIND=true`.

Manually Specify Include, Library Location and Links

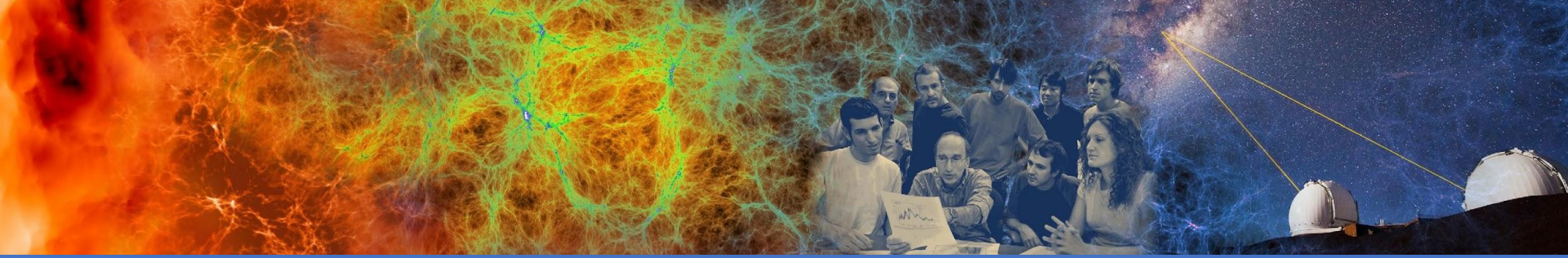
```
epalmer> # In this example, we will show how to manually include and link libraries during
 the compile step. The example code I will use, requires
```

Manually Specify Include, Library Location and Links



List of commands from previous slide

```
1.  ls
2.  module list
3.  cc hypre_ex.c -o hypre_ex
4.  export HYPRE_DIR=/global/u1/e/elvis/NERSC_User_Training/hypre/src/hypre
5.  ls $HYPRE_DIR/lib
6.  ls $HYPRE_DIR
7.  cc hypre_ex.c -I${HYPRE_DIR}/include  -L${HYPRE_DIR}/lib -o hypre_ex
8.  srun -n 1 -c 2 ./hypre_ex
```



Understanding Job Parameters

Guiding Example Batch Job Script

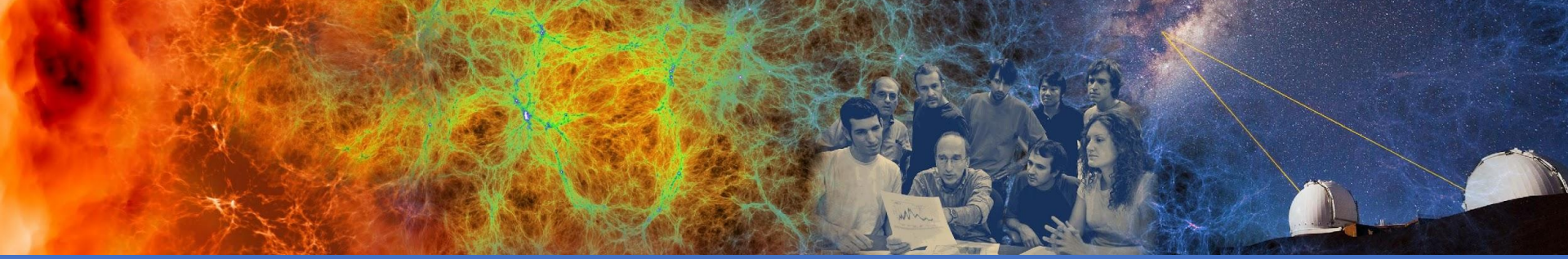
```
#!/bin/bash

#SBATCH -N 2
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH --mail-user=elvis@lbl.gov
#SBATCH -t 0:30:00

export OMP_NUM_THREADS=8
export OMP_PLACES=threads
export OMP_PROC_BIND=spread


srun -n 32 -c 16 --cpu_bind=cores ./demo_code
```

Key Terms:

- Node
- MPI Task
- *Logical* CPU
- Thread
- Physical Core
- Processor

Advanced Terms:

- NUMA Domain

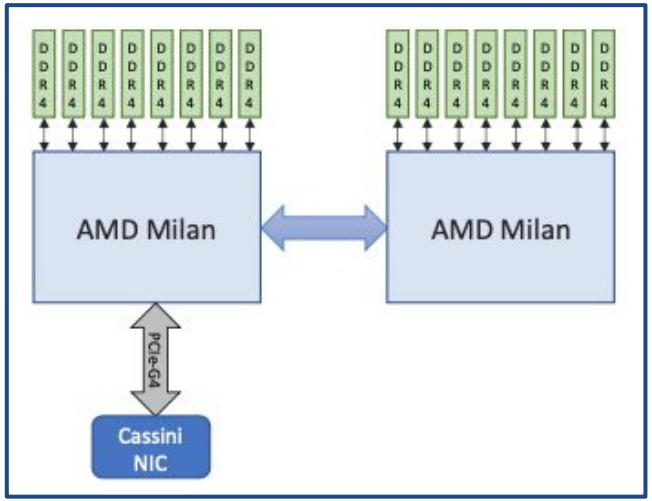


Understanding Job Parameters:
Hardware: Node, Processor, Physical Core, Logical CPU

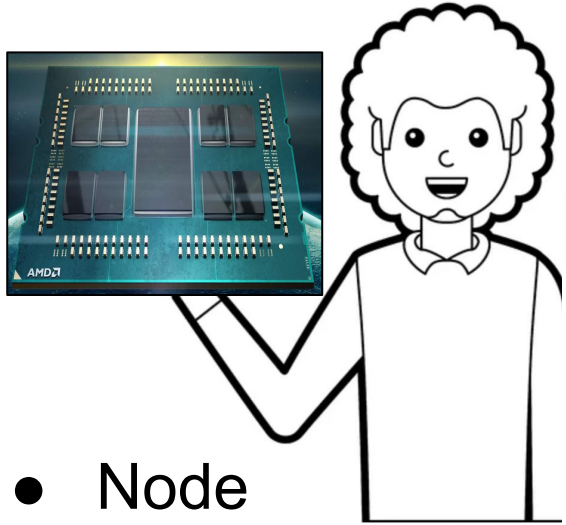
Perlmutter CPU Node Terms

From Perlmutter system architecture	This Talk
2x AMD EPYC 7763 (Milan) CPUs	2x AMD EPYC 7763 (Milan) Processors
64 Cores per CPU	64 Physical Cores per processor
2 Hyperthreads per core	2 Logical CPUs per physical core
4 NUMA domains per socket	4 NUMA domains per processor

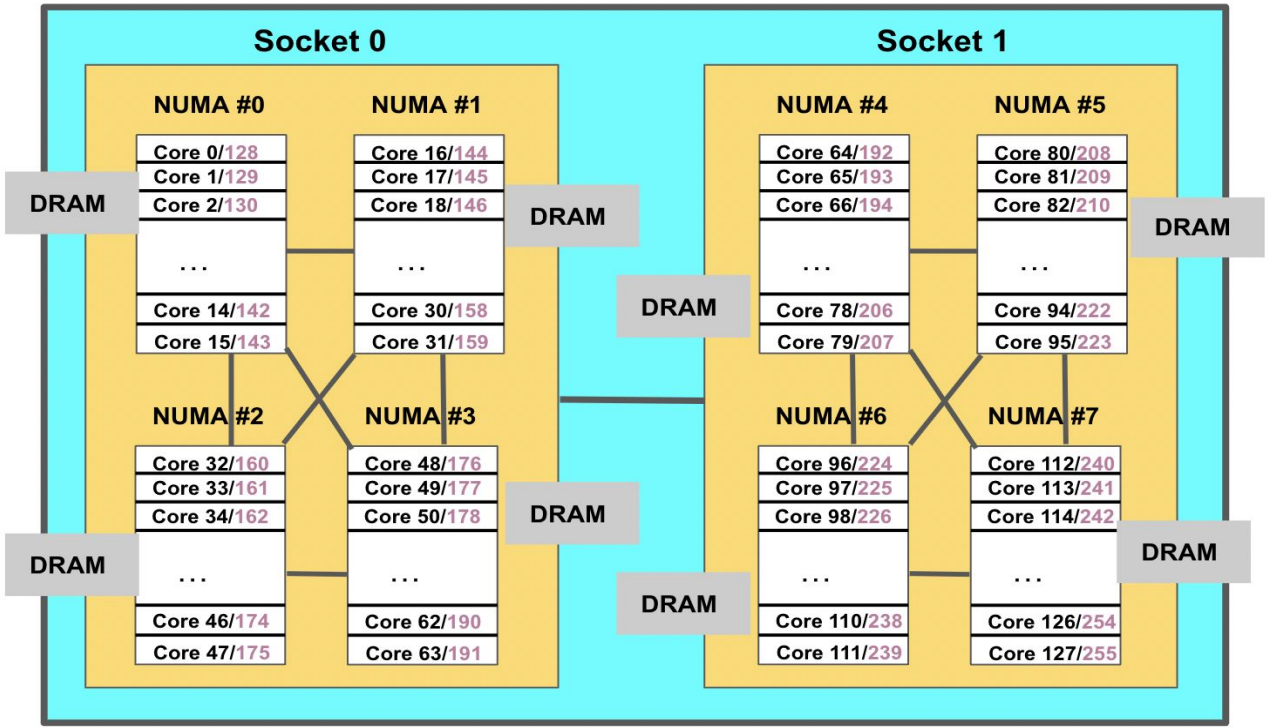
Diagram of Perlmutter CPU Node



Perlmutter CPU Compute Node



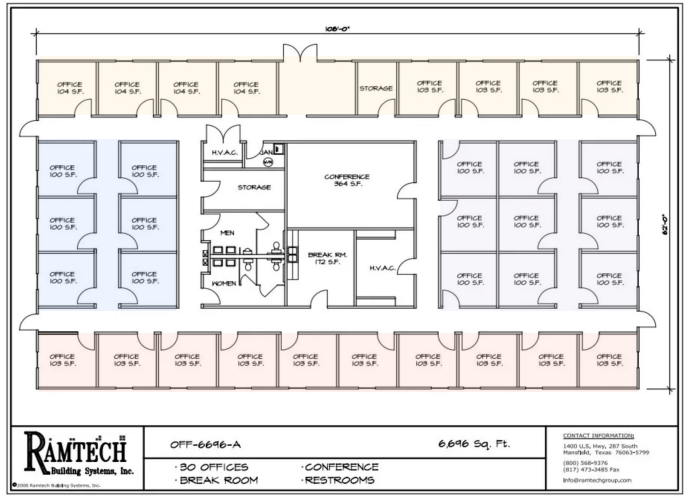
- Node
- Processor
- Physical Core
- *Logical* CPU



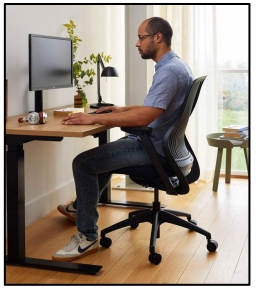
Office Building Analogy for Node Architecture

Compute Node(s)

Physical Cores



Processor



Logical CPUs / Hardware Threads

NERSC

BERKELEY LAB
Bringing Science Solutions to the World

U.S. DEPARTMENT OF ENERGY | Office of Science

Sample Batch Job Script

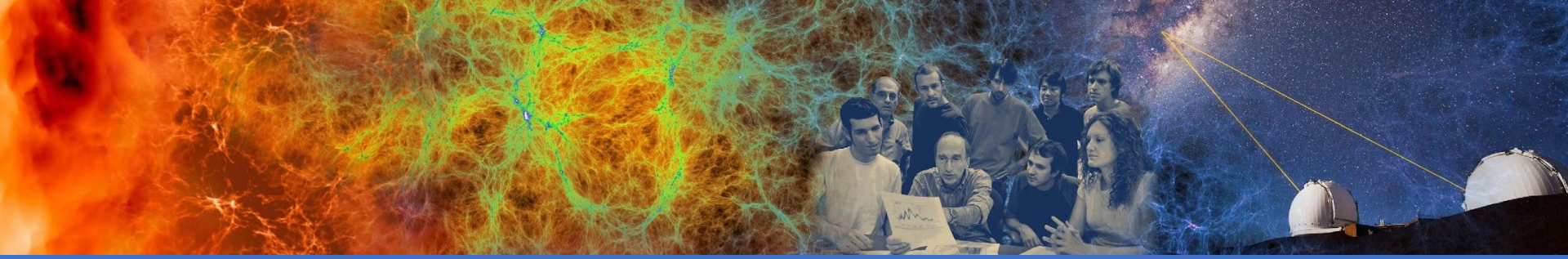
```
#!/bin/bash

#SBATCH -N 2
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH --mail-user=elvis@lbl.gov
#SBATCH -t 0:30:00

export OMP_NUM_THREADS=8
export OMP_PLACES=threads
export OMP_PROC_BIND=spread


srun -n 32 -c 16 --cpu_bind=cores ./demo_code
```

Key Terms:

- Node
- MPI Task
- *Logical* CPU
- Thread
- Physical Core
- Processor

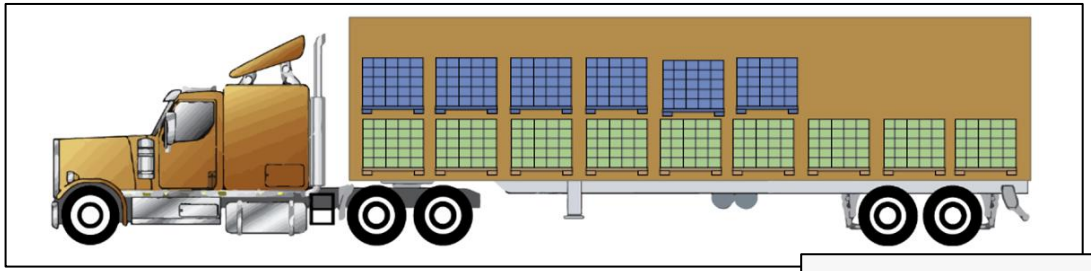
Advanced Terms:

- NUMA Domain



Understanding Job Parameters:
Splitting Up Work: MPI Tasks, (OpenMP) Threads

Cargo Analogy for MPI Tasks & OMP Threads



Simulation Code

MPI Task

(OpenMP) Thread

Guiding Example Batch Job Script

```
#!/bin/bash

#SBATCH -N 2
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH --mail-user=elvis@lbl.gov
#SBATCH -t 0:30:00

export OMP_NUM_THREADS=8
export OMP_PLACES=threads
export OMP_PROC_BIND=spread


srun -n 32 -c 16 --cpu_bind=cores ./demo_code
```

Key Terms:

- Node
- MPI Task
- *Logical* CPU
- Thread
- Physical Core
- Processor

Advanced Terms:

- NUMA Domain

Settings to Address NUMA Performance

- Use `--cpu_bind=cores` when
  the # of MPI tasks ≤ the # of physical cores
- Use `--cpu_bind=threads` when
  the # of MPI tasks > the # of physical cores
- In hybrid MPI/OpenMP code, use at least 8 MPI tasks to avoid NUMA penalties when using OpenMP threads
- The value of `-c` should be ≥ the value of `OMP_NUM_THREADS`
- For thread affinity set:

```
OMP_PROC_BIND=spread
OMP_PLACES=threads
```

Sample Batch Job Script

```
#!/bin/bash

#SBATCH -N 2
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH --mail-user=elvis@lbl.gov
#SBATCH -t 0:30:00

export OMP_NUM_THREADS=8
export OMP_PLACES=threads
export OMP_PROC_BIND=spread


srun -n 32 -c 16 --cpu_bind=cores ./demo_code
```

Key Terms:

- Node
- MPI Task
- *Logical* CPU
- Thread
- Physical Core
- Processor

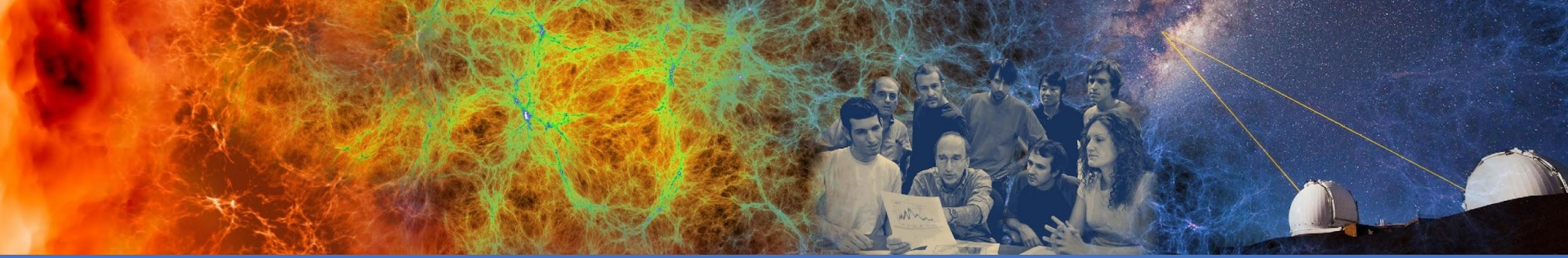
Advanced Terms:

- NUMA Domain

Compute Nodes Comparison for CPU Affinity

	Cori Haswell	Cori KNL	Perlmutter CPU	CPU on Perlmutter GPU
Physical cores	32	68	128	64
Logical CPUs per physical core	2	4	2	2
Logical CPUs per node	64	272	256	128
NUMA domains	2	1	8	4
-c value for srun	2·⌊32/t.p.n.⌋	4·⌊68/t.p.n.⌋, but 4·⌊64/t.p.n.⌋ recommended	2·⌊128/t.p.n.⌋	2·⌊64/t.p.n.⌋

t.p.n. = Number of MPI **t**asks **p**er **n**ode $\lfloor x \rfloor = \text{floor}(x)$



Job Scripts

Sample Batch Job Script – MPI Only

Cori Haswell

```
#!/bin/bash

#SBATCH -N 40
#SBATCH -C haswell
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=1

srun -n 1280 -c 2 \
--cpu_bind=cores  ./demo_code
```

Perlmutter CPU

```
#!/bin/bash

#SBATCH -N 10
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=1

srun -n 1280 -c 2 \
--cpu_bind=cores  ./demo_code
```

Tip: As a best practice for MPI only runs, include `export OMP_NUM_THREADS=1`

Sample Batch Job Script – MPI Only

Cori Haswell

```
#!/bin/bash

#SBATCH -N 40
#SBATCH -C haswell
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=1

srun -n 1280 -c 2 \
--cpu_bind=cores  ./demo_code
```

1280/40=32

2·⌊32/32⌋=2

Perlmutter CPU

```
#!/bin/bash

#SBATCH -N 10
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=1

srun -n 1280 -c 2 \
--cpu_bind=cores  ./demo_code
```

1280/10=128

2·⌊128/128⌋=2

Tip: As a best practice for MPI only runs, include `export OMP_NUM_THREADS=1`

Sample Batch Job Script – MPI Only

Cori Haswell

```
#!/bin/bash

#SBATCH -N 40
#SBATCH -C haswell
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=1

srun -n 1280 -c 2 \
--cpu_bind=cores  ./demo_code
```

Perlmutter CPU

```
#!/bin/bash

#SBATCH -N 40
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=1

srun -n 1280 -c 8 \
--cpu_bind=cores  ./demo_code
```

Tip: As a best practice for MPI only runs, include `export OMP_NUM_THREADS=1`

Sample Batch Job Script – MPI Only

Cori Haswell

```
#!/bin/bash

#SBATCH -N 40
#SBATCH -C haswell
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=1

srun -n 1280 -c 2 \
--cpu_bind=cores  ./demo_code
```

1280/40=32

2·⌊32/32⌋=2

Perlmutter CPU

```
#!/bin/bash

#SBATCH -N 40
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=1

srun -n 1280 -c 8 \
--cpu_bind=cores  ./demo_code
```

1280/40=32

2·⌊128/32⌋=8

Tip: As a best practice for MPI only runs, include `export OMP_NUM_THREADS=1`

Sample Batch Job Script – MPI & OpenMP

Question

```
#!/bin/bash

#SBATCH -N 32
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=8
export OMP_PROC_BIND=spread
export OMP_PLACES=threads

srun -n 512 -c ?  --cpu_bind=cores ./demo_code
```

What should the value of `-c` be?

Sample Batch Job Script – MPI & OpenMP

Question

```
#!/bin/bash

#SBATCH -N 32
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=8
export OMP_PROC_BIND=spread
export OMP_PLACES=threads

srun -n 512 -c ? \
--cpu_bind=cores  ./demo_code
```

Hint

$512 / 32 = 16$

$2 \cdot \lfloor 128 / 16 \rfloor = 16$

$16 \geq 8$

Considered because using OpenMP

Sample Batch Job Script – MPI & OpenMP

Question

```
#!/bin/bash

#SBATCH -N 32
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=8
export OMP_PROC_BIND=spread
export OMP_PLACES=threads

srun -n 512 -c ? \
--cpu_bind=cores  ./demo_code
```

Process

1. MPI tasks / Nodes

   $512 \;/\; 32 = 16$

2. Logical CPUs / MPI tasks

   $2 \cdot \lfloor 128 \;/\; 16 \rfloor = 16$

3. Check logical CPUs greater than threads

   $16 \geq 8$

Considered because using OpenMP

Sample Batch Job Script – MPI & OpenMP

Question

```
#!/bin/bash

#SBATCH -N 32
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=8
export OMP_PROC_BIND=spread
export OMP_PLACES=threads

srun -n 512 -c ? \
--cpu_bind=cores  ./demo_code
```

Answer

```
#!/bin/bash

#SBATCH -N 32
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=8
export OMP_PROC_BIND=spread
export OMP_PLACES=threads

srun -n 512 -c 16 \
--cpu_bind=cores  ./demo_code
```

NERSC Job Script Generator

https://iris.nersc.gov/jobscript • https://my.nersc.gov/script_generator.php

Jobscript Generator

This tool generates a batch script template which also realizes specific process and thread binding configurations.

Configuration:

Machine

Select the machine on which you want to submit your job.

Perlmutter - CPU

Application Name

Specify your application including the full path.

my_app

Job Name

Specify a name for your job.

test_job1

Email Address

Specify your email address to get notified when the job enters a certain state.

elvis@lbl.gov

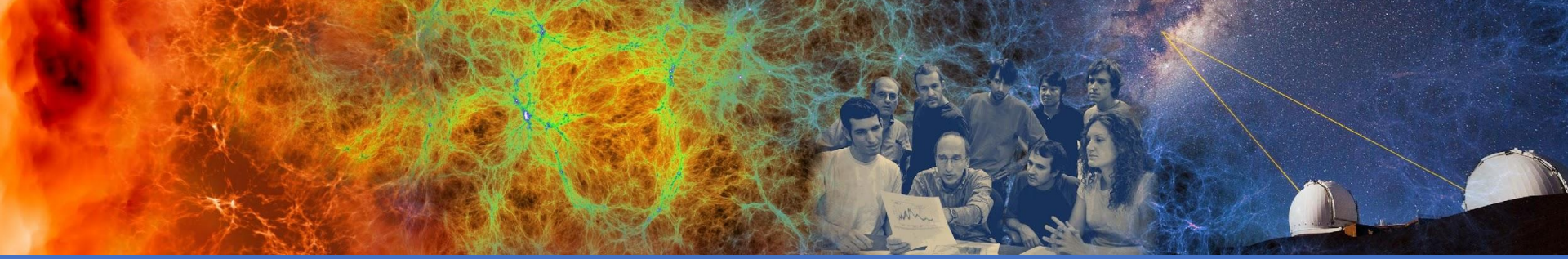
Quality of Service

Select the QoS you request for your job.

Job script:

```
#!/bin/bash

# Base script generated by NERSC Batch Script Generator on
https://iris.nersc.gov/jobscript

#SBATCH -N 8
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J test_job1
#SBATCH --mail-user=elvis@lbl.gov
#SBATCH --mail-type=ALL
#SBATCH -t 0:30:0

# OpenMP settings:
export OMP_NUM_THREADS=1
export OMP_PLACES=threads
export OMP_PROC_BIND=true

#run the application:
srun -n 128 -c 16 --cpu_bind=cores  my_app
```

This should be:
export OMP_PROC_BIND=spread

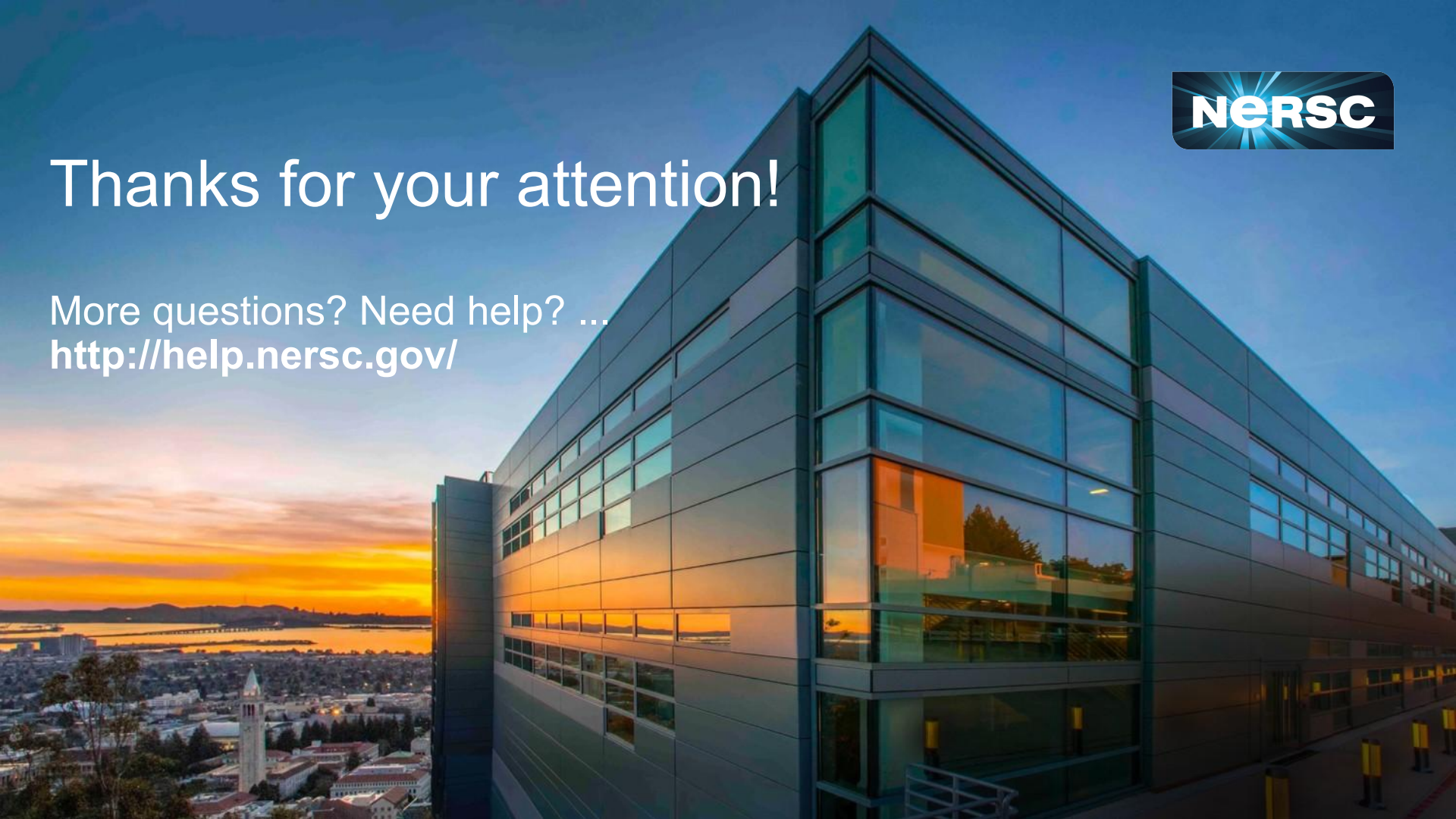


Key Suggestions:

- Use `module spider` for comprehensive module search
- Recompile Cori code with `PrgEnv-gnu`, `PrgEnv-cray`, or PrgEnv-nvidia
- Use the compiler wrappers
- Recalculate job script parameters for optimal performance

CPU Hands-on Exercises

- Feel free to use some NERSC prepared CPU examples at
  - https://github.com/NERSC/Migrate-to-Perlmutter/tree/main/CPU
  - or bring your own applications codes today.
- Follow README.first and README for each example
  - hello-example: serial and MPI
  - matrix-example (C) or jacobi-example (Fortran): hybrid MPI/OpenMP
  - xthi-example: affinity
  - gsl_test: using package available from E4S stack
- Perlmutter Compute node reservations, 11:30 - 14:30:
  - CPU: #SBATCH --reservation=pm_cpu_mar10 -A ntrain8 -C cpu
  - Existing NERSC users are added to the ntrain2 project to access node reservations



Thanks for your attention!

More questions? Need help? ...
http://help.nersc.gov/